The facilitator is responsible for planning, conducting and following up the session (meeting). In that regard he must invite people to a session (and create user IDs for the ones that do not have access to the electronic collaborative work environment) and check to see that everything is clear. During the session, the facilitator conducts the session (moves through the agenda), and helps with questions. After the session, the facilitator has a responsibility of following up the session with regard to tasks and participants' wishes. Amongst these responsibilities is distributing the report, although everyone participating may create one at any time. Present invention supports these processes.

Inviting people: present invention has its own user database which contains information about the user, this includes:

Name (first and last)
License holder (who is paying for the use of the system) – which will be tied up to a pricing and billing system.
Department, Location, Phone numbers
Email-address (which also enables users to use the system as a stand-alone mail-program).

The facilitator may only invite people who does not have access to the system through the invitation system. If he needs to invite people who do not yet have access to the program, then he must contact the system administrator who is able to create a user profile for the ones the facilitator needs to be invited. This process is very quick, and includes sending out a mail with user ID and password (and installation files).

The facilitator conducts the session by starting and stopping activities (participants may not contribute to an activity that is not active). In addition the participant may send pop-up chat messages to get everyone's attention whilst using present

invention. The participants generally click on an "OK"-button for the message-box to disappear.

There are two types of system administrators: client company administrator (CCA) and super-administrator (SA). The client company administrator is responsible for the use of the present invention within the client organisation. Thus client employees who need access to the system contact the CCA to get the installation file and basic support. In addition, the CCA will be responsible to update and maintain the current user list from that specific client company. The CCA is also able to create company groups. He is not able to view or edit other companies' users. The CCA is able to designate roles

The SA is able to create client company administrators and designate roles (administrator, normal user or guest user) for any company. He does not have access to specific sessions (due to security issues). The SA is able to support the CCA with more advanced support questions.

These roles are supported by present invention. By screening the information sent out to the CCA, present invention enables a secure environment for several companies to work on the same server. In addition, the SA has privileges which enable him to support the various processes.

In one embodiment, the present invention provides collaborative service over the Internet, to make a ubiquitous collaborative work environment. Figure **1A** is an illustration showing a ubiquitous collaborative work environment **120** in accordance with an embodiment of the present invention. The ubiquitous collaborative work environment **120** includes a physical collaborative work environment **122** having users **124**, and an electronic collaborative work environment **126** having electronic agents **128** and collaborative software **130**.

The electronic collaborative work environment utilizes the key elements of people, knowledge, services and supporting software. The ability to instantly share knowledge, search for information and work together will allow members of the collaborative work environment to co-operate on a new level. The collaborative work environment can be split into a physical component (people, knowledge services) - CWE and an electronic component (supporting software).

Figure **1B** is an illustration showing an electronic collaborative work environment **126**, in accordance with an embodiment of the present invention. The electronic collaborative work environment **126** includes electronic agents **128** and collaborative software **130**.

The electronic agents **128** act as intermediaries between the collaborative software **130** and the users **124**. In addition each electronic agent serves as an intermediary between the user **124** and the electronic agent **128** of other users **124**. electronic agents **128** aid users **124** in the search for others users **124** with the same interests and competencies. As such they are a part of the foundation to create virtual communities, and assist the user **124** in tasks. To illustrate with an example: The electronic agents **128** are able to search out users **124** with wanted competencies and other attributes and allow a user **124** through a directory service to pick the users **124** that comply with their requirements. These can then be invited to the user's personalised electronic collaborative work environment **126**.

The electronic agent **128** also enables information gathering both within and outside the collaborative environment, by searching for information/persons that might be of interest to the owner of the electronic agent. As such, extensive profiles may be utilised, to create a more proficient electronic agent. The more information the user enters about himself, the better equipped will the electronic agent **128** be to search and communicate with other agents – and thus provide for a dynamic and vibrant community of users. The electronic agents aid the users to meet the right people at the right time.

Conventional meeting facilitating software ranges across a wide field of asynchronous, synchronous, distributed and undistributed forms of virtual collaboration. The problem with conventional meeting facilitating software is that they are not made for supporting the meeting process.

A preferred embodiment of a system in accordance with the present invention is preferably practiced in the context of a personal computer such as an IBM compatible personal computer, Apple Macintosh computer or UNIX based workstation. A representative hardware environment is depicted in Figure **2**, which illustrates a typical hardware configuration of a workstation in accordance with a preferred embodiment having a central processing unit **210**, such as a microprocessor, and a number of other units interconnected via a system bus **212**. The workstation shown in Figure **2** includes a Random Access Memory (RAM) **214**, Read Only Memory (ROM) **216**, an I/O adapter **218** for connecting peripheral devices such as disk storage units **220** to the bus **212**, a user interface adapter **222** for connecting a keyboard **224**, a mouse **226**, a speaker **228**, a microphone **232**, and/or other user interface devices such as a touch screen (not shown) to the bus **212**, communication adapter **234** for connecting the workstation to a communication network (e.g., a data processing network) and a display adapter **236** for connecting the bus **212** to a display device **238**. The workstation typically has resident thereon an operating system such as the Microsoft Windows NT or Windows/95 Operating System (OS), the IBM OS/2 operating system, the MAC OS, or UNIX operating system. Those skilled in the art will appreciate that the present invention may also be implemented on platforms and operating systems other than those mentioned.

A preferred embodiment is written using JAVA, C, and the C++ language and utilizes object oriented programming methodology. Object oriented programming (OOP) has become increasingly used to develop complex applications. As OOP moves toward the mainstream of software design and development, various software solutions require adaptation to make use of the benefits of OOP. A need exists for these principles of OOP to be applied to a messaging interface of an electronic messaging system such that a set of OOP classes and objects for the messaging interface can be provided.

OOP is a process of developing computer software using objects, including the steps of analyzing the problem, designing the system, and constructing the program. An object is a software package that contains both data and a collection of related structures and procedures. Since it contains both data and a collection of structures and procedures, it can be visualized as a self-sufficient component that does not require other additional structures, procedures or data to perform its specific task. OOP, therefore, views a computer program as a collection of largely autonomous components, called objects, each of which is responsible for a specific task. This concept of packaging data, structures, and procedures together in one component or module is called encapsulation.

In general, OOP components are reusable software modules which present an interface that conforms to an object model and which are accessed at run-time through a component integration architecture. A component integration architecture is a set of architecture mechanisms which allow software modules in different process spaces to utilize each others capabilities or functions. This is generally done by assuming a common component object model on which to build the architecture. It is worthwhile to differentiate between an object and a class of objects at this point. An object is a single instance of the class of objects, which is often just called a class. A class of objects can be viewed as a blueprint, from which many objects can be formed.

OOP allows the programmer to create an object that is a part of another object. For example, the object representing a piston engine is said to have a composition-relationship with the object representing a piston. In reality, a piston engine comprises a piston, valves and many other components; the fact that a piston is an element of a piston engine can be logically and semantically represented in OOP by two objects.

OOP also allows creation of an object that "depends from" another object. If there are two objects, one representing a piston engine and the other representing a piston engine wherein the piston is made of ceramic, then the relationship between the two objects is not that of composition. A ceramic piston engine does not make up a piston engine. Rather it is merely one kind of piston engine that has one more limitation than the piston engine; its piston is made of ceramic. In this case, the object representing the ceramic piston engine is called a derived object, and it inherits all of the aspects of the object representing the piston engine and adds further limitation or detail to it. The object representing the ceramic piston engine "depends from" the object representing the piston engine. The relationship between these objects is called inheritance.

When the object or class representing the ceramic piston engine inherits all of the aspects of the objects representing the piston engine, it inherits the thermal characteristics of a standard piston defined in the piston engine class. However, the ceramic piston engine object overrides these ceramic specific thermal characteristics, which are typically different from those associated with a metal piston. It skips over the original and uses new functions related to ceramic pistons. Different kinds of piston engines have different characteristics, but may have the same underlying functions associated with it (e.g., how many pistons in the engine, ignition sequences, lubrication, etc.). To access each of these functions in any piston engine object, a programmer would call the same functions with the same names, but each type of piston engine may have different/overriding implementations of functions behind the same name. This ability to hide different implementations of a

function behind the same name is called polymorphism and it greatly simplifies communication among objects.

With the concepts of composition-relationship, encapsulation, inheritance and polymorphism, an object can represent just about anything in the real world. In fact, one's logical perception of the reality is the only limit on determining the kinds of things that can become objects in object-oriented software. Some typical categories are as follows:

- Objects can represent physical objects, such as automobiles in a traffic-flow simulation, electrical components in a circuit-design program, countries in an economics model, or aircraft in an air-traffic-control system.
- Objects can represent elements of the computer-user environment such as windows, menus or graphics objects.
- An object can represent an inventory, such as a personnel file or a table of the latitudes and longitudes of cities.
- An object can represent user-defined data types such as time, angles, and complex numbers, or points on the plane.

With this enormous capability of an object to represent just about any logically separable matters, OOP allows the software developer to design and implement a computer program that is a model of some aspects of reality, whether that reality is a physical entity, a process, a system, or a composition of matter. Since the object can represent anything, the software developer can create an object which can be used as a component in a larger software project in the future.

If 90% of a new OOP software program consists of proven, existing components made from preexisting reusable objects, then only the remaining 10% of the new software project has to be written and tested from scratch. Since 90% already came from an inventory of extensively tested reusable objects, the potential domain from which an error could originate is 10% of the program. As a result, OOP enables software developers to build objects out of other, previously built objects.

This process closely resembles complex machinery being built out of assemblies and sub-assemblies. OOP technology, therefore, makes software engineering more like hardware engineering in that software is built from existing components, which are available to the developer as objects. All this adds up to an improved quality of the software as well as an increased speed of its development.

Programming languages are beginning to fully support the OOP principles, such as encapsulation, inheritance, polymorphism, and composition-relationship. With the advent of the C++ language, many commercial software developers have embraced OOP. C++ is an OOP language that offers a fast, machine-executable code. Furthermore, C++ is suitable for both commercial-application and systems-programming projects. For now, C++ appears to be the most popular choice among many OOP programmers, but there is a host of other OOP languages, such as Smalltalk, Common Lisp Object System (CLOS), and Eiffel. Additionally, OOP capabilities are being added to more traditional popular computer programming languages such as Pascal.

The benefits of object classes can be summarized, as follows:

- Objects and their corresponding classes break down complex programming problems into many smaller, simpler problems.
- Encapsulation enforces data abstraction through the organization of data into small, independent objects that can communicate with each other. Encapsulation protects the data in an object from accidental damage, but allows other objects to interact with that data by calling the object's member functions and structures.
- Subclassing and inheritance make it possible to extend and modify objects through deriving new kinds of objects from the standard classes available in the system. Thus, new capabilities are created without having to start from scratch.

- Polymorphism and multiple inheritance make it possible for different programmers to mix and match characteristics of many different classes and create specialized objects that can still work with related objects in predictable ways.
- Class hierarchies and containment hierarchies provide a flexible mechanism for modeling real-world objects and the relationships among them.
- Libraries of reusable classes are useful in many situations, but they also have some limitations. For example:
- Complexity. In a complex system, the class hierarchies for related classes can become extremely confusing, with many dozens or even hundreds of classes.
- Flow of control. A program written with the aid of class libraries is still responsible for the flow of control (i.e., it must control the interactions among all the objects created from a particular library). The programmer has to decide which functions to call at what times for which kinds of objects.
- Duplication of effort. Although class libraries allow programmers to use and reuse many small pieces of code, each programmer puts those pieces together in a different way. Two different programmers can use the same set of class libraries to write two programs that do exactly the same thing but whose internal structure (i.e., design) may be quite different, depending on hundreds of small decisions each programmer makes along the way. Inevitably, similar pieces of code end up doing similar things in slightly different ways and do not work as well together as they should.

Class libraries are very flexible. As programs grow more complex, more programmers are forced to reinvent basic solutions to basic problems over and over again. A relatively new extension of the class library concept is to have a framework of class libraries. This framework is more complex and consists of significant collections of collaborating classes that capture both the small scale patterns and major mechanisms that implement the common requirements and design in a specific application domain. They were first developed to free

application programmers from the chores involved in displaying menus, windows, dialog boxes, and other standard user interface elements for personal computers.

Frameworks also represent a change in the way programmers think about the interaction between the code they write and code written by others. In the early days of procedural programming, the programmer called libraries provided by the operating system to perform certain tasks, but basically the program executed down the page from start to finish, and the programmer was solely responsible for the flow of control. This was appropriate for printing out paychecks, calculating a mathematical table, or solving other problems with a program that executed in just one way.

The development of graphical user interfaces began to turn this procedural programming arrangement inside out. These interfaces allow the user, rather than program logic, to drive the program and decide when certain actions should be performed. Today, most personal computer software accomplishes this by means of an event loop which monitors the mouse, keyboard, and other sources of external events and calls the appropriate parts of the programmer's code according to actions that the user performs. The programmer no longer determines the order in which events occur. Instead, a program is divided into separate pieces that are called at unpredictable times and in an unpredictable order. By relinquishing control in this way to users, the developer creates a program that is much easier to use. Nevertheless, individual pieces of the program written by the developer still call libraries provided by the operating system to accomplish certain tasks, and the programmer must still determine the flow of control within each piece after it's called by the event loop. Application code still "sits on top of" the system.

Even event loop programs require programmers to write a lot of code that should not need to be written separately for every application. The concept of an application framework carries the event loop concept further. Instead of dealing with all the nuts and bolts of constructing basic menus, windows, and dialog boxes and then

making these things all work together, programmers using application frameworks start with working application code and basic user interface elements in place. Subsequently, they build from there by replacing some of the generic capabilities of the framework with the specific capabilities of the intended application.

Application frameworks reduce the total amount of code that a programmer has to write from scratch. However, because the framework is really a generic application that displays windows, supports copy and paste, and so on, the programmer can also relinquish control to a greater degree than event loop programs permit. The framework code takes care of almost all event handling and flow of control, and the programmer's code is called only when the framework needs it (e.g., to create or manipulate a proprietary data structure).

A programmer writing a framework program not only relinquishes control to the user (as is also true for event loop programs), but also relinquishes the detailed flow of control within the program to the framework. This approach allows the creation of more complex systems that work together in interesting ways, as opposed to isolated programs, having custom code, being created over and over again for similar problems.

Thus, as is explained above, a framework basically is a collection of cooperating classes that make up a reusable design solution for a given problem domain. It typically includes objects that provide default behavior (e.g., for menus and windows), and programmers use it by inheriting some of that default behavior and overriding other behavior so that the framework calls application code at the appropriate times.

There are three main differences between frameworks and class libraries:

- Behavior versus protocol. Class libraries are essentially collections of behaviors that you can call when you want those individual behaviors in your program. A framework, on the other hand, provides not only behavior but

also the protocol or set of rules that govern the ways in which behaviors can be combined, including rules for what a programmer is supposed to provide versus what the framework provides.

- Call versus override. With a class library, the code the programmer instantiates objects and calls their member functions. It's possible to instantiate and call objects in the same way with a framework (i.e., to treat the framework as a class library), but to take full advantage of a framework's reusable design, a programmer typically writes code that overrides and is called by the framework. The framework manages the flow of control among its objects. Writing a program involves dividing responsibilities among the various pieces of software that are called by the framework rather than specifying how the different pieces should work together.
- Implementation versus design. With class libraries, programmers reuse only implementations, whereas with frameworks, they reuse design. A framework embodies the way a family of related programs or pieces of software work. It represents a generic design solution that can be adapted to a variety of specific problems in a given domain. For example, a single framework can embody the way a user interface works, even though two different user interfaces created with the same framework might solve quite different interface problems.

Thus, through the development of frameworks for solutions to various problems and programming tasks, significant reductions in the design and development effort for software can be achieved. A preferred embodiment of the invention utilizes HyperText Markup Language (HTML) to implement documents on the Internet together with a general-purpose secure communication protocol for a transport medium between the client and ? . HTTP or other protocols could be readily substituted for HTML without undue experimentation. Information on these products is available in T. Berners-Lee, D. Connoly, "RFC 1866: Hypertext Markup Language - 2.0" (Nov. 1995); and R. Fielding, H, Frystyk, T. Berners-Lee, J. Gettys and J.C. Mogul, "Hypertext Transfer Protocol -- HTTP/1.1: HTTP Working Group

Internet Draft" (May 2, 1996). HTML is a simple data format used to create hypertext documents that are portable from one platform to another. HTML documents are SGML documents with generic semantics that are appropriate for representing information from a wide range of domains. HTML has been in use by the World-Wide Web global information initiative since 1990. HTML is an application of ISO Standard 8879; 1986 Information Processing Text and Office Systems; Standard Generalized Markup Language (SGML).

To date, Web development tools have been limited in their ability to create dynamic Web applications which span from client to server and interoperate with existing computing resources. Until recently, HTML has been the dominant technology used in development of Web-based solutions. However, HTML has proven to be inadequate in the following areas:

- Poor performance;
- Restricted user interface capabilities;
- Can only produce static Web pages;
- Lack of interoperability with existing applications and data; and
- Inability to scale.

Sun Microsystem's Java language solves many of the client-side problems by:

- Improving performance on the client side;
- Enabling the creation of dynamic, real-time Web applications; and
- Providing the ability to create a wide variety of user interface components.

With Java, developers can create robust User Interface (UI) components. Custom "widgets" (e.g., real-time stock tickers, animated icons, etc.) can be created, and client-side performance is improved. Unlike HTML, Java supports the notion of client-side validation, offloading appropriate processing onto the client for improved performance. Dynamic, real-time Web pages can be created. Using the above-mentioned custom UI components, dynamic Web pages can also be created.

Sun's Java language has emerged as an industry-recognized language for "programming the Internet." Sun defines Java as: "a simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multithreaded, dynamic, buzzword-compliant, general-purpose programming language. Java supports programming for the Internet in the form of platform-independent Java applets." Java applets are small, specialized applications that comply with Sun's Java Application Programming Interface (API) allowing developers to add "interactive content" to Web documents (e.g., simple animations, page adornments, basic games, etc.). Applets execute within a Java-compatible browser (e.g., Netscape Navigator) by copying code from the server to client. From a language standpoint, Java's core feature set is based on C++. Sun's Java literature states that Java is basically, "C++ with extensions from Objective C for more dynamic method resolution."

Another technology that provides similar function to JAVA is provided by Microsoft and ActiveX Technologies, to give developers and Web designers wherewithal to build dynamic content for the Internet and personal computers. ActiveX includes tools for developing animation, 3-D virtual reality, video and other multimedia content. The tools use Internet standards, work on multiple platforms, and are being supported by over 100 companies. The group's building blocks are called ActiveX Controls, small, fast components that enable developers to embed parts of software in hypertext markup language (HTML) pages. ActiveX Controls work with a variety of programming languages including Microsoft Visual C++, Borland Delphi, Microsoft Visual Basic programming system and, in the future, Microsoft's development tool for Java, code named "Jakarta." ActiveX Technologies also includes ActiveX Server Framework, allowing developers to create server applications. One of ordinary skill in the art readily recognizes that ActiveX could be substituted for JAVA without undue experimentation to practice the invention.

Figure **3** is a flowchart illustrating a method **300** for affording collaboration planning in a collaborative work tool architecture, in accordance with an embodiment of the present invention. First, in operation **302**, a client user interface including an activity data field is provided. Then, in operation **304**, activity data is received from a facilitator user, wherein the activity data includes a start time for the activity and a duration of the activity. The received activity data is then stored on a server via a network, as indicated in operation **306**. Finally, a plurality of participant users are allowed access to the stored activity data via the network. See operation **308**.

In one aspect of the present invention the participant users asynchronously access the activity data. In another aspect, the participant users synchronously access the activity data.

In one embodiment of the present innovation, the collaborative work tool architecture affords non-distributed work groups. In another embodiment, the collaborative work tool architecture affords distributed work groups.

Additionally, the client user interface may enable real-time user discussion utilizing a chat window.

When planning a meeting utilizing the present invention, a facilitator user is able to invite participant users to a session using the client user interface, discussed in greater detail subsequently. The facilitator user is then able to generate a list of activities, which is similar to an agenda, to occur during the session. The activities can be defined using at least six different activity tools, which are essentially collaboration techniques. These activity tools include, brainstorming tools, discussion tools, categorization tools, voting tools, action list (summary) tools, and external activity (breaks) tools.

Figure **4** is a flowchart illustrating a method **400** for listing activities in a graphical user interface in a collaborative work tool framework, in accordance with one

embodiment of the present invention. In an initial operation **402**, an activity window having activity start data, activity duration data, and an activity status data is displayed. Then, an activity is defined in response to user selection of a addactivity button, wherein the defined activity is thereafter displayed in the activity window as shown in operation **404**. Finally, in operation **406**, a status for the defined activity is determined based on activity start data for the defined activity and activity duration data for the defined activity.

In one aspect of the present invention, the activity may be defined as a brainstorming activity. Alternatively, the activity may be defined as a discussion activity. Optionally, the activity may be defined as a categorization activity. Also optionally, the activity may be defined as a voting activity. In addition, the activity may be defined as a summary activity.

In one embodiment of the present invention, a message window capable of displaying user messages in real-time may be displayed Additionally, the defined data may be sent to a specific participant user in response to user selection of the specific participant user from a participant user menu. . In another embodiment, the defined activity may be sent to a database in response to user selection of a submit button.

Figure **5** is an illustration showing a graphical user interface **500** for listing activities in a collaborative work tool framework, in accordance with one embodiment of the present invention. The graphical user interface **500** includes a list of activities **502**, wherein each activity includes an activity start time **504**, an activity duration **506**, an activity title **508**, and an activity status **510**.

The graphical user interface **500** further includes a list of predefined activity types **512**, a real-time message window **514**, and a participant menu **516**. The user is further able to define the start time for the activity **504**, the duration of the activity

506, and the title of the activity **508** using the list of activities **502** area of the graphical user interface **502**.

In this manner, an activity list for a session may be created. Furthermore, other session participants may view the agenda in preparation for the session. Participants may further interact with one another using the real-time message window **514**. The participant menu **516** may be selected using a computer pointing device. Once selected, the participant menu preferable displays a list of session participants to which a user may send messages and other information. Thus, using the participant menu **516**, a user may direct communications to specific users, or choose to send communications to a group of users. Optionally, the participants users may use the participant menu ("Online Now") **520** to direct communications to specific users. Finally, during the session, the present invention determines a status **510** for each activity in the list of activities **502** utilizing the activity start time **504** and the activity duration **506**.

Figure **6** is a flowchart showing a method **600** for conducting activities in a collaborative work tool architecture, in accordance with an embodiment of the present invention. In an initial operation **602**, a client user interface including at least one activity data field is afforded. Then, in operation **604**, a selection of a particular activity data field from a user is received. Additional information on the selected activity field is then presented using the client user interface. See operation **606**. The user is then allowed to input data concerning the selected activity data field as indicated in operation **608**. In operation **610**, the received data is stored on a server via a network. Finally, in operation **612**, a plurality of participant users are allowed access to the stored data via the network.

In one aspect of the present invention, the meeting data may be voting data on a predefined topic. Optionally, the meeting data may be user readable sentences concerning a predefined topic. Alternatively, the data may be user readable assigned tasks as defined by the participant users.

In one embodiment of the present invention, the data may be accessed by the participant users asynchronously. In another embodiment, the data may be accessed by the participant users synchronously. In one embodiment of the present innovation, the collaborative work tool architecture affords non-distributed work groups. In another embodiment, the collaborative work tool architecture affords distributed work groups.

The present invention allows a user to conduct a session having formal collaboration, using the activity tools, and/or informal communication, using the real-time message window. Moreover, the present invention allows the user to conduct multiple activities within a session simultaneously. Additionally, the user may conduct activities anonymously utilizing the present invention.

Figure **7** is an illustration showing a graphical user interface **700** for conducting sessions in a collaborative work tool architecture, in accordance with an embodiment of the present invention. The graphical user interface **700** includes session selection tabs **702**, a list of sessions **704**, wherein each session **704** includes a list of sessions **706**. As discussed previously, each session **706** includes an activity start date **708**, a activity start time **710**, asession duration **712**, an activity title **714**, and an activity status **716**. In addition, a facilitator **718** is listed for each activity.

In use, a user may elect to participate in a session **704** by selecting a reply button **720** using a computer selection device, such as a mouse. In this manner, users may participate in multiple sessions simultaneously. Moreover, individual users may selectively indicate which sessions they will participate in and facilitator users may select which users to invite to particular sessions. As discussed in greater detail previously, the present invention determines a status **716** for each activity as the sessions proceed.

Figure **7A** is a flowchart showing a method **1** for brainstorming in a collaborative work tool architecture, in accordance with an embodiment of the present invention. In operation **2**, a single-tier discussion tree is displayed in response to selection of a brainstorming button by a facilitator user. Then, tree node text providing tree node definitions is received from the user wherein the user is able to contribute to the single-tier discussion tree, as indicated in operation **4**. The tree node definitions are then stored in a database using a network. See operation **6**. Finally, in operation **8**, access to the tree node definitions is provided to a plurality of participant users, wherein the participant users are able to contribute to the single-tier discussion tree, as discussed in greater detail with reference to Figure **7B** next.

Figure **7B** is an illustration showing a graphical user interface **721** for performing brainstorming activities in a collaborative work tool framework, in accordance with one embodiment of the present invention. The graphical user interface **721** includes an edit button **722**, a view button **724**, a list of predefined activity types **726**, a real-time message window **728**, a participant menu **730**, a brainstorming contribution window **732** having idea expressions **734**, and a contribution input widow **736**.

In use, the user is further able to add "brainstorming" ideas to the list of idea expressions **734** utilizing the contribution input window **736**. The user enters their contribution into the contribution input window **736** and then submits the entered idea expression utilizing a submit button **738**. The entered idea expression is then be listed in the brainstorming contribution window **732**. It should be borne in mind that "brainstorming" in the present invention generally allows only posting. This is, generally replies to idea contributions are not allowed in the brainstorming activity. For replies, the discussion activity is utilized, as discussed in greater detail subsequently.

Figure **7C** is a flowchart showing a method **740** for providing discussion in a collaborative work tool architecture, in accordance with an embodiment of the present invention. In operation **742**, a single-tier discussion tree is displayed in